(FILE 'USPAT' ENTERED AT 15:16:16 ON 16 AUG 1997)

```
L1          264 S ZERO-EXTEN?
L2          131 S 395/705/CLS
L3            3 S L1 AND L2
L4            0 S ENTENDED (3A) IMMEDIATE
L5          139 S EXTENDED (3A) IMMEDIATE
L6           30 S L1 AND L5
L7            0 S L6 AND L2
L8           28 S L1 (P) L5
L9          865 S SIGN# (A) EXTEN?
L10          28 S L8 AND L9
L11           7 S L8 (P) L9
```

=> d cit 1-

1. 5,603,047, Feb. 11, 1997, Superscalar microprocessor architecture; Robert L. Caulk, Jr., 395/800 :IMAGE AVAILABLE:

2. 5,568,630, Oct. 22, 1996, Backward-compatible computer architecture with extended word size and address space; Earl A. Killian, et al., 395/376; 364/DIG.1; 395/380, 413, 800 :IMAGE AVAILABLE:

3. 5,481,684, Jan. 2, 1996, Emulating operating system calls in an alternate instruction set using a modified code segment descriptor; David E. Richter, et al., 395/388; 364/DIG.1; 395/500, 568 :IMAGE AVAILABLE:

4. 5,420,992, May 30, 1995, Backward-compatible computer architecture with extended word size and address space; Earl A. Killian, et al., 395/800; 364/240.3, 258.1, 259.5, DIG.1, DIG.2; 395/500 :IMAGE AVAILABLE:

5. 5,388,841, Feb. 14, 1995, External memory system having programmable graphics processor for use in a video game system or the like; Jeremy E. San, et al., 463/44; 345/122; 395/118, 501, 502, 508 :IMAGE AVAILABLE:

6. 5,357,604, Oct. 18, 1994, Graphics processor with enhanced memory control circuitry for use in a video game system or the like; Jeremy E. San, et al., 463/31; 345/24, 189; 395/501, 508; 463/44 :IMAGE AVAILABLE:

7. 5,177,701, Jan. 5, 1993, Computer and method for performing immediate calculation by utilizing the computer; Shigeaki Iwasa, 364/736 :IMAGE AVAILABLE:

(FILE 'USPAT' ENTERED AT 15:16:16 ON 16 AUG 1997)
L1 264 S ZERO-EXTEN?
L2 131 S 395/705/CLS
L3 3 S L1 AND L2
L4 0 S ENTENDED (3A) IMMEDIATE
L5 139 S EXTENDED (3A) IMMEDIATE
L6 30 S L1 AND L5
L7 0 S L6 AND L2
L8 28 S L1 (P) L5
L9 865 S SIGN# (A) EXTEN?

=> s l8 and l9
L10 28 L8 AND L9

=> s l8 (p) l9
L11 7 L8 (P) L9

=> d cit,ab,kwic 1-

1. 5,603,047, Feb. 11, 1997, Superscalar microprocessor architecture; Robert L. Caulk, Jr., 395/800 :IMAGE AVAILABLE:

US PAT NO: 5,603,047 :IMAGE AVAILABLE: L11: 1 of 7

ABSTRACT:
A microprocessor core operating on instructions in a dual six-stage pipeline. Instructions are fetched and decoded by an instruction scheduling unit which includes a queuing stage for facilitating conditional branch operations. Instructions can be executed in five execution units including a load/store/add unit, an ALU unit, a shift/multiply unit, a branch unit, and a coprocessor which interfaces with the microprocessor core. Exceptions are handled by the coprocessor which includes a plurality of registers and a multiple entry translation lookaside buffer and an exception program counter. When an exception is detected the coprocessor loads the exception program counter with a restart address where execution can resume after the exception is serviced, the plurality of registers being used during the exception processing. One of the registers is a circulate mask register which is used by the coprocessor in executing an Add with Circular Mask instruction in which an immediate field of the instruction is sign-extended and added to the contents of a general register, the result being masked with the extended value in the circular mask register.

DETDESC:

DETD(138)
. . .
Instruction
Format and Description

ALU IMMEDIATE INSTRUCTION SUMMARY
Add Immediate
ADDI rt, rs, immediate
Add 16-bit, **sign-extended immediate** to register rs and place 32-bit result in register rt. Trap on two's complement overflow.
Add Immediate

ADDIU rt, rs, immediate
Unsigned Add 16-bit, **sign-extended immediate** to register rs and place 32-bit result in register rt. Do not trap on overflow.

Set on Less Than Immediate
SLTI rt, rs, immediate
Compare 16-bit, **sign-extended immediate** with register rs as unsigned 32-bit integers. Result = 1 if rs is less than immediate; otherwise result = 0. Place result in register rt.

Set of Less Than Immediate Unsigned
SLTIU rt, rs, immediate
Compare 16-bit, **sign-extended immediate** with register rs as unsigned 32-bit integers. Result = 1 if rs is less than immediate; otherwise result = 0. Place result in register rt.

AND Immediate
ANDI rt, rs, immediate
Zero-extend 16-bit immediate, AND with contents of register rs, and place result in register rt.

OR Immediate
ORI rt, rs, immediate
Zero-extend 16-bit immediate, OR with contents of register rs, and place result in register rt.

Exclusive OR Immediate
XORI rt, rs, immediate
Zero-extend 16-bit immediate, exclusive OR with contents of register rs, and place result in register rt.

Load Upper . . . rd.

Shift Right Arithmethic
SRA, rd, rt, shamt
Shift contents of register rt right by shamt bits, **sign-extending** the high-order bits. Place 32-bit result in register rd.

Shift Left Logical Variable
SLLV rd, rt, rs
. . . of register rt right. Low-order Variable 5 bits of register rs specify the number of bits to shift. **Sign-extend** the high-order bits of rt and place 32-bit result in register rd.

MULTIPLY/DIVIDE INSTRUCTION SUMMARY

Multiply MULT rs,. . . to special register LO.

EXTENSIONS SUMMARY (CW4010 ISA EXTENSIONS)

Add Circular Immediate
ADDCIU rt, rs, immediate
The 16-bit **immediate** is **sign extended** and added to the contents of general register rs, with the result masked by the value in

. . .

2. 5,568,630, Oct. 22, 1996, Backward-compatible computer architecture with extended word size and address space; Earl A. Killian, et al., 395/376; 364/DIG.1; 395/380, 413, 800 :IMAGE AVAILABLE:

US PAT NO: 5,568,630 :IMAGE AVAILABLE: L11: 2 of 7

ABSTRACT:
A technique for extend[redacted] the data word size and the vi[redacted]al address space of a pre-existing architecture so that hardware for the extended architecture also supports the pre-existing architecture. Extension of the data word size from m bits to N bits entails widening the machine registers and data paths from m bits to N bits and sign-extending entities of m or fewer bits to N bits when they are loaded into registers. Some of the m-bit instructions, when operating on N-bit sign-extended versions of m-bit entities, produce an N-bit result that may not correspond to the correct m-bit result, sign-extended to N bits. For these instructions compatibility requires that the instructions be further defined to guarantee a sign-extended result. This means that separate N-bit instructions corresponding to these m-bit instructions are needed. The support for up to an N-bit virtual address space is provided in part by widening the virtual address data paths. The extended architecture supports the m-bit architecture's addressing with minimal additional hardware. This is made possible by storing m-bit addresses as N-bit entities in sign-extended form and requiring that the results of address computations on these entities be in sign-extended form.

DETDESC:

DETD(69)

Table 3C sets forth the ALU immediate instructions. The bit-wise logical immediate instructions (ANDI, ORI, and XORI) produce **sign-extended** results when presented with **sign-extended** input operands. It is noted that the 16-bit **immediate** value is **zero-extended** to 32 bits in the 32-bit architecture and to 64 bits in the 64-bit architecture before being combined with the. . .

3. 5,481,684, Jan. 2, 1996, Emulating operating system calls in an alternate instruction set using a modified code segment descriptor; David E. Richter, et al., 395/388; 364/DIG.1; 395/500, 568 :IMAGE AVAILABLE:

US PAT NO: 5,481,684 :IMAGE AVAILABLE: L11: 3 of 7

ABSTRACT:
The CISC architecture is extended to provide for segments that can hold RISC code rather than just CISC code. These new RISC code segments have descriptors that are almost identical to the CISC segment descriptors, and therefore these RISC descriptors may reside in the CISC descriptor tables. The global descriptor table in particular may have CISC code segment descriptors for parts of the operating system that are written in x86 CISC code, while also having RISC code segment descriptors for other parts of the operating system that are written in RISC code. An undefined or reserved bit within the descriptor is used to indicate which instruction set the code in the segment is written in. An existing user program may be written in CISC code, but call a service routine in an operating system that is written in RISC code. Thus existing CISC programs may be executed on a processor that emulates a CISC operating system using RISC code. A processor capable of decoding both the CISC and RISC instruction sets is employed. The switch from CISC to RISC instruction decoding is triggered when control is transferred to a new segment, and the segment descriptor indicates that the code within the segment is written in the alternate instruction set.
SYSTEM LIMITS EXCEEDED - DISPLAY ENDED

=> d 4- cit,ab,kwic

4. 5,420,992, May 30, 1995, Backward-compatible computer architecture with extended word size and address space; Earl A. Killian, et al., 395/800; 364/240.3, 258.1, 259.5, DIG.1, DIG.2; 395/500 :IMAGE AVAILABLE:

US PAT NO: 5,420,992 :IMAGE AVAILABLE: L11: 4 of 7

ABSTRACT:
A technique for extending the data word size and the virtual address space of a pre-existing architecture so that hardware for the extended architecture also supports the pre-existing architecture. Extension of the data word size from m bits to N bits entails widening the machine registers and data paths from m bits to N bits and sign-extending entities of m or fewer bits to N bits when they are loaded into registers. Some of the m-bit instructions, when operating on N-bit sign-extended versions of m-bit entities, produce an N-bit result that may not correspond to the correct m-bit result, sign-extended to N bits. For these instructions compatibility requires that the instructions be further defined to guarantee a sign-extended result. This means that separate N-bit instructions corresponding to these m-bit instructions are needed. The support for up to an N-bit virtual address space is provided in part by widening the virtual address data paths. The extended architecture supports the m-bit architecture's addressing with minimal additional hardware. This is made possible by storing m-bit addresses as N-bit entities in sign-extended form and requiring that the results of address computations on these entities be in sign-extended form.

DETDESC:

DETD(69)

Table 3C sets forth the ALU immediate instructions. The bit-wise logical immediate instructions (ANDI, ORI, and XORI) produce **sign-extended** results when presented with **sign-extended** input operands. It is noted that the 16-bit **immediate** value is **zero-extended** to 32 bits in the 32-bit architecture and to 64 bits in the 64-bit architecture before being combined with the. . .

5. 5,388,841, Feb. 14, 1995, External memory system having programmable graphics processor for use in a video game system or the like; Jeremy E. San, et al., 463/44; 345/122; 395/118, 501, 502, 508 :IMAGE AVAILABLE:

US PAT NO: 5,388,841 :IMAGE AVAILABLE: L11: 5 of 7

ABSTRACT:
A fully programmable, graphics microprocessor is embodied in a removable external memory unit for connection with a host information processing system. In an exemplary embodiment, a video game system is described including a host video game system and a pluggable video game cartridge housing the graphics microprocessor. The game cartridge also includes a read-only program memory (ROM) and a random-access memory (RAM). The graphics coprocessor operates in conjunction with a three bus architecture embodied on the game cartridge. The graphics processor using this bus architecture may execute programs from either the program ROM, external RAM or its own internal cache RAM. The fully user programmable graphics coprocessor has an instruction set which is designed to efficiently implement arithmetic operations associated with 3-D graphics and, for example, includes special instructions executed by dedicated hardware for plotting individual pixels in the host video game system's character mapped display which, from the programmer's point of view, creates a "virtual" bit map by permitting the addressing of individual pixels--even though the host system is character based. The graphics coprocessor interacts with the host coprocessor such that the graphics coprocessor's 16 general registers are accessible to the host processor at all times.

DETDESC:

DETD(153)
. . .

```
                    Link return address to r11
                    r11 = r15 + 1 . . . 4
$95:    SEX         Sign extend low byte to word
                    DReg.:b15-b7: = SReg.:b7:
                    DReg.1 = SReg.1
$96     ASR         Arithmetic shift right
                    DReg =. . .
                    c = (SReg) * r6).b15
$a0-$af nn
        IBT r0 . . . r15, sbyte
                    Load rn with sign extended byte
                    rn = immediate byte (sign extended)
if alt1:
        LMS r0 . . . r15, byte
                    Load rn from absolute shifted byte address
                    rn=RAM:byte<<1:. . . 1
                    (TO/WITH/FROM ignored)
$ef     GETB        Get unsigned byte from ROM buffer to Dreg
                    DReg = ROM buffer byte. zero extended
if alt1:
        GETBH       Get from ROM buffer to high byte of Dreg
                    DReg = ROM buffer byte. merge. . . (use WITH)
if alt1+alt2:
        GETBS       Get signed byte from ROM buffer to Dreg
                    DReg = ROM buffer byte. sign extended
$f0-$ff nnnn
        IWT r0 . . . r15, word
                    Load immediate word to rn
                    rn = immediate. . .
```

6. 5,357,604, Oct. 18, 1994, Graphics processor with enhanced memory control circuitry for use in a video game system or the like; Jeremy E. San, et al., 463/31; 345/24, 189; 395/501, 508; 463/44 :IMAGE AVAILABLE:

US PAT NO: 5,357,604 :IMAGE AVAILABLE: L11: 6 of 7

ABSTRACT:
A fully programmable, graphics microprocessor is embodied in a removable external memory unit for connection with a host information processing system. In an exemplary embodiment, a video game system is described including a host video game system and a pluggable video game cartridge housing the graphics microprocessor. The game cartridge also includes a read-only program memory (ROM) and a random-access memory (RAM). The graphics coprocessor operates in conjunction with a three bus architecture embodied on the game cartridge. The graphics processor using this bus architecture may execute programs from either the program ROM, external RAM or its own internal cache RAM. The fully user programmable graphics coprocessor has an instruction set which is designed to efficiently implement arithmetic operations associated with 3-D graphics and, for example, includes special instructions executed by dedicated hardware for plotting individual pixels in the host video game system's character mapped display which, from the programmer's point of view, creates a "virtual" bit map by permitting the addressing of individual pixels--even though the host system is character based. The graphics coprocessor interacts with the host coprocessor such that the graphics coprocessor's 16 general registers are accessible to the host processor at all times.

DETDESC:

DETD(151)
. . .
. 4

```
                    Link return address to r11
                    r11 = r15 + 1 . . . 4
```

```
$95:    SEX         Sign extend low byte to word
                    DReg.b15-b7: = SReg.:b7:
                    DReg.1 = SReg.1
$96     ASR         ARithmetic shift right
                    DReg =.  .  .
                    c = (SReg) * r6).b15
$a0-$af nn
        IBT r0 . . . r15, sbyte
                    Load rn with sign extended byte
                    rn = immediate byte (sign extended)
if alt1:
        LMS r0 . . . r15, byte
                    Load rn from absolute shifted byte address
                    rn=RAM:byte<<1:(word.  .  .  -1
                    (TO/WITH/FROM ignored)
$ef     GETB        Get unsigned byte from ROM buffer to Dreg
                    DReg = ROM buffer byte. zero extended
if alt1:
        GETBH       Get from ROM buffer to high byte of Dreg
                    DReg = ROM buffer byte. merge.  .  .  (use WITH)
if alt1+alt2:
        GETBS       Get signed byte from ROM buffer to Dreg
                    DReg = ROM buffer byte, sign extended
$f0-$ff nnnn
        IWT r0 . . . r15, word
                    Load immediate word to rn
                    rn = immediate.  .  .
```

7. 5,177,701, Jan. 5, 1993, Computer and method for performing immediate calculation by utilizing the computer; Shigeaki Iwasa, 364/736 :IMAGE AVAILABLE:

US PAT NO: 5,177,701 :IMAGE AVAILABLE: L11: 7 of 7

ABSTRACT:
A computer for performing immediate calculations by executing an immediate calculation instruction containing a first immediate value and an immediate prefixed instruction containing a second immediate value consists of a register for storing the second immediate value, a prefix state flag for setting a flag in cases where the second immediate value is stored in the register, concatenating unit for concatenating the first immediate value with the second immediate value in cases where the flag in the prefix state flag is set and generating the concatenated immediate value as a first constant, bit extension unit for extending the number of bits in the first immediate value without changing the first immediate value and generating the extended immediate value as a second constant, selector for selecting the first constant generated in the concatenating unit in cases where the flag in the prefix state flag is set and selecting the second constant generated in the bit extension unit in cases where the flag in the prefix state flag is not set, and calculation unit for performing the immediate calculation to process the constant selected in the selector.

SUMMARY:

BSUM(11)

Therefore, . . . is smaller than the number of bits required for the calculation, the number of bits in the field for the **immediate** operand is **extended** to a required bit number to execute the immediate calculation instruction by a **sign extender** or a **zero extender** in which a required number of "0"s is merely added to the high-order.

SUMMARY:

BSUM(53)

an **extended immediate** selector for selecting the first immediate value added the prescribed bits by the **sign extension** unit as the second constant in cases where the instruction interpretation means interprets the first immediate value to have the sign, and for selecting the first immediate value added the bits by the **zero extension** unit as the second constant in cases where the instruction interpretation means interprets the first immediate value to have no. . .

DETDESC:

DETD(50)

an **extended immediate** selector 54 for selecting the 32-bit **immediate** operand **extended** by the **sign extension** unit 52 in cases where the 12-bit immediate operand in the immediate field assigned to the immediate calculation instruction has a sign, and for selecting the 32-bit **immediate** operand **extended** by the **zero extension** unit 53 in cases where the 12-bit immediate operand in the immediate field assigned to the immediate calculation instruction has. . . where the flag in the prefixed state flag 50 is set and, for selecting the 32-bit constant selected by the **extended immediate** selector 54 in cases where the flag in the prefixed state flag 50 is reset.

CLAIMS:

CLMS(4)

4. . . . calculation instruction stored in the instruction registering means has a sign, and
the bit extension and constant generating means comprises:
a **zero extension** unit for adding "0" bits to the high-order of the first immediate value contained in the calculation instruction;
a **sign extension** unit for adding "0" or "1" bits to the high-order of the first immediate value contained in the calculation instruction according to the sign of the first **immediate** value; and
an **extended immediate** selector for
(1) selecting the first immediate value added the prescribed bits by the **sign extension** unit as the second constant in cases where the instruction interpretation means interprets the first immediate value to have the sign and
(2) selecting the first immediate value added the bits by the **zero extension** unit as the second constant in cases where the instruction interpretation means interprets the first immediate value to have no. . .

```
     (FILE 'USPAT' ENTERED AT 12:09:00 ON 16 AUG 1997)
                E (SUZUKI, MASATO)/IN
L1            34 S E3
                E (KAMIYAMA, HIROSHI)/IN
L2             3 S E3
                E (MIYAJI, SHINYA)/IN
L3        159526 S CUSTOM? OR (USER (2A) (SPECIF? OR DESIGNAT?))
L4         12224 S (PROGRAM# OR SOFTWARE OR CODE) (2A) (TRANSLAT? OR CONVER
?)
L5          2889 S L4 AND L3
L6            61 S POINTER# (A) (WIDTH# OR SIZE#)
L7             0 S L5 AND L6
L8           302 S L4 (P) L3
L9             5 S L8 (P) POINTER#
L10          574 S (POINTER# OR ADDRESS##) (A) (WIDTH# OR SIZE#)
L11         3797 S DATA (A) (WIDTH# OR SIZE#)
L12           45 S L10 (P) L11
L13            9 S L12 AND L4
L14           12 S L12 AND L3
L15            1 S L14 AND L4
L16        13644 S (PROGRAM# OR SOFTWARE OR CODE OR INSTRUCTION#) (2A) (TRA
NSL
L17            4 S L6 AND L16
L18            4 S L17 AND L10
L19            4 S L18 AND L11
L20          881 S COMPIL?/TI,AB
L21            2 S L6 AND L20
L22          131 S 395/705/CLS
L23            2 S L6 AND L22
L24            2 S L21 OR L23
L25            0 S L22 AND L11 AND L6
L26            0 S L22 AND L11 AND L10
L27           51 S L22 AND L3
L28            0 S L27 AND L6
L29            0 S L27 AND L10

=> d l22 cit,ab,kwic
TEXT DATA FOR PATENT 5,659,753 IS NOT AVAILABLE, SEE IMAGE DATA, THE
MICROFILE OR PAPER INSTEAD

=> d l23 cit,ab,kwic

1.  5,617,569, Apr. 1, 1997, Method and system for implementing pointers
to members in a compiler for an object-oriented programming language; Jan
Gray, et al., 395/614; 364/283.1, DIG.1; 395/705 :IMAGE AVAILABLE:

US PAT NO:     5,617,569 :IMAGE AVAILABLE:            L23: 1 of 2

ABSTRACT:
A method and system in an object-oriented environment for determining the
offset of a data member of a derived class when the derived class has a
virtually inherited base class and the data member is defined in the base
class. In a preferred embodiment, the base class has a data structure and
a class address. The base class data structure has the data member
located at a data member offset from the base class address. The derived
class has a data structure and a class address. The derived class data
structure includes an occurrence of the base class data structure. The
occurrence of the base class data structure is located at a base class
```

offset from the derived class address. In a preferred embodiment, the base class offset is stored at a memory location, a first field and a second field for a pointer to the data member are allocated, the data member offset is stored in the first field, an indicator of the stored base class offset is stored in the second field, the pointer is dereferenced by retrieving the base class offset using the indicator stored in the second field, retrieving the data member offset from the first field, and adding the retrieved base class offset to the retrieved data member offset to give the offset of the data member within the derived data structure.
US-CL-CURRENT: 395/614; 364/283.1, DIG.1; **395/705**

DETDESC:

DETD(138)

FIG. . . . field is qdisp, which contains the offset of the start of the offsets of the virtually inherited bases. However, the **pointer size** is not bounded in space for the same reasons as discussed above with respect to FIG. 6.

=> d l17 cit,ab,kwic 1,2

1. 5,581,721, Dec. 3, 1996, Data processing unit which can access more registers than the registers indicated by the register fields in an instruction; Hideo Wada, et al., 395/376, 412 :IMAGE AVAILABLE:

US PAT NO: 5,581,721 :IMAGE AVAILABLE: L17: 1 of 4

ABSTRACT:
The data processing unit includes a greater number of physical floating point registers than the number of floating point registers accessible by an instruction, window start point register having a plurality of bits, 1-bit window start pointer valid register, conversion apparatus for converting a floating point register number in an instruction to a physical floating point register number when the value of the window start pointer valid register is 1, and changing the pattern of this conversion by a value obtained from the value of the window start pointer register or the value of a window stride designated in a specific instruction, and the value of the window start pointer register. Also provided is an instruction controller for detecting a window start pointer set instruction for setting a value to the window start pointer register, a floating point register pre-load **instruction** for **converting** the floating point register number in the instruction to a physical floating point register number by the conversion circuit from the value obtained from the value of the window start pointer register and the value of the window stride, and storing a main memory data in the physical floating point register indicated by the physical floating point register number.

ABSTRACT:
The . . . window start pointer set instruction for setting a value to the window start pointer register, a floating point register pre-load **instruction** for **converting** the floating point register number in the instruction to a physical floating point register number by the conversion circuit from. . .

SUMMARY:

BSUM(47)

When . . . window start pointer valid register is 1 in the floating point register preload instruction and the floating point register post-store **instruction**, **conversion** of the logical floating point register number to the physical floating point register number is

effected as described above, and. . .

SUMMARY:

BSUM(50)

Hereinafter, sm will be called a "**pointer width** maximum value".

SUMMARY:

BSUM(79)
3. The window cut width can be changed by changing the **pointer width** maximum value sm or the window start pointer register width q.

SUMMARY:

BSUM(114)

Furthermore, . . . width and the register number slide quantity can be freely changed by changing the window pointer register width (generally, the **pointer width** maximum width is fixed), and for this reason, flexible programming can be made.

DETDESC:

DETD(16)

According to the system of the present invention, therefore, a certain floating point register number in the **instruction** can be **converted** to a different physical floating point register number by changing the window start pointer and the window stride in the. . .

CLAIMS:

CLMS(4)

4. . . .
a window start pointer register width, a certain integer value greater than said window start pointer register width, as a **pointer width** maximum value, the power of 2 using, as an index, the value obtained by subtracting said window start pointer register width from said **pointer width** maximum value, as a window cut width, and the quotient obtained by dividing said number of registers by said window. . . cut width, as the number of windows, during conversion from the register number to said physical register number is the **instruction** by said **conversion** means,
said register number in said register pre-load **instruction** is **converted** to said physical register number expressed by the sum obtained by adding said register number in said instruction to a. . . window stride value in the instruction by using said window number as a modulus,
said register number in said register post-store **instruction** is **converted** to said physical register number expressed by said sum of the register number in the instruction and a value of. . . using said window number as a modulus and
said register number in said load instruction, said store instruction and said arithmetic **instruction** is **converted** to said physical register number expressed by the sum of said register number in the instruction and a value of. . .

CLAIMS:

CLMS(5)

5. A data processing unit according to claim 4, wherein said window cut width is made variable by rendering said **pointer width** maximum value or said window start pointer register width variable.

CLAIMS:

CLMS(7)

7. . . . processing unit according to claim 4, wherein the number of said physical registers is the power of 2 using said **pointer width** maximum value as an index.

CLAIMS:

CLMS(8)

8. A data processing unit according to claim 7, wherein said window cut width is made variable by rendering said **pointer width** maximum value or said window start pointer register width variable.

CLAIMS:

CLMS(10)

10. . . . a window start pointer register width; a certain integer value greater than said window start pointer register width, as a **pointer width** maximum value; the power of 2 using, as an index, the difference obtained by subtracting said window start pointer register width from said **pointer width** maximum value, as a window cut width; and the quotient obtained by dividing said local register number by said window. . .

CLAIMS:

CLMS(13)

13. . . . processing unit according to claim 10, wherein the number of said physical registers is the power of 2 using said **pointer width** maximum value as an index.

CLAIMS:

CLMS(14)

14. A data processing unit according to claim 13, wherein said window cut width is made variable by rendering said **pointer width** maximum value or said window start pointer register width variable.

2. 5,201,039, Apr. 6, 1993, Multiple address-space data processor with addressable register and context switching; Ken Sakamura, 395/411; 364/230, 230.2, 231.8, 232.23, 239, 240, 243, 243.4, 243.41, 246, 246.1, 246.11, 246.3, 247, 247.7, 251, 254, 254.5, 254.6, 258, 258.1, 258.2, 258.3, 259, 259.5, 259.7, 265, 270.5, 280, 280.4, 926.1, 926.3, 926.9, 926.92, 927.2, 927.4, 927.81, 927.92, 927.93, 931, 931.4, 931.49, 933, 933.2, 933.3, 933.4, 933.6, 933.7, 937.1, 937.2, 937.4, 938, 938.1, 938.4, 941, 942.7, 943.9, 944.92, 946.2, 946.9, 947, 947.2, 947.6, 948.3, 948.34, 950, 955, 955.5, 955.6, 961.2, 961.3, 964, 964.2, 965, 965.4, 966.1, 966.4, 966.5, 973, 977, DIG.1, DIG.2; 395/412, 569 :IMAGE AVAILABLE:

US PAT NO: 5,201,039 :IMAGE AVAILABLE: L17: 2 of 4

ABSTRACT:
Two or more address spaces are provided in a data processor. One of the address spaces comprises control registers so that the control registers

can be accessed using instructions having an address in the second address space. High-speed context switching can be accomplished by allotting the context-saving area to the second address space. The context can be saved in various formats specified by a context format register.

DETDESC:

DETD(135)

Normally, . . . executes an object code for the data processor 32 of the present invention, it provides a mode which changes the **pointer size** to 32 bits. Since this mode is specified in PSW, it is possible to use a 32-bit type program and. . .

DETDESC:

DETD(138)

It . . . difficult to use pointers which differ in size, because they serve to identify the location. If there is a 64-bit **size pointer** together with a 32-bit **size pointer**, the location cannot be identified unless the size of all the pointers in 64 bits. Therefore, even if a 32-bit. . .

DETDESC:

DETD(3136)
This . . . instructions, the instruction is reexecuted as a page out exception (POE). In addition, while referencing the ATE with the ACS **instruction**, an address **translation** exception (ATRE) or bus access exception (BAE) may occur.

DETDESC:

DETD(3314)

If PSTLB is executed at AT=00, the effective address of prgaddr is calculated without an address **translation** like other **instructions**. However, the instruction operation of PSTLB does not depend on the value of AT. In other words, even if AT=00,. . .

DETDESC:

DETD(3601)

The . . . requires MMU, until the execution environment concerning MMU (such as page table) is terminated, it is necessary to execute the **instructions** without address **translation**.

3. 5,630,083, May 13, 1997, Decoder for decoding multiple instructions in parallel; Adrian L. Carbine, et al., 395/388; 364/259.9, 262.4, 262.7, 262.8, DIG.1; 395/387 :IMAGE AVAILABLE:

US PAT NO: 5,630,083 :IMAGE AVAILABLE: L13: 3 of 9

ABSTRACT:
A decoder for decoding multiple instructions in parallel, including a full decoder that can decode an instruction into multiple micro-operations, and a partial decoder that can decode a subset of the full instruction set.

SUMMARY:

BSUM(6)

Computers . . . executable by the computer processor. In order to run these high level programs, the program is compiled by a compiler **program** that **translates** the higher level instructions into macroinstructions having a format that can be decoded and executed. The compiled macroinstructions are supplied. . .

DETDESC:

DETD(31)

As . . . result of extracted data, architectural machine state, or some combination of the two. Examples of alias fields include logical registers, **address size**, **data size**, stack address and stack **data size**, immediate and displacement data, branch information, and portions of various predetermined opcodes.

DETDESC:

DETD(32)

The . . . base field 272, a register index field 274, an immediate data register 276, a displacement data register 278, a stack **address size** field 280, a **data size** field 284 which indicates the size of the data as well as some floating point information, and an **address size** field 286 that indicates whether the data address is 16- or 32-bits.

4. 5,566,298, Oct. 15, 1996, Method for state recovery during assist and restart in a decoder having an alias mechanism; Darrell D. Boggs, et al., 395/182.08, 182.13, 185.02 :IMAGE AVAILABLE:

US PAT NO: 5,566,298 :IMAGE AVAILABLE: L13: 4 of 9

ABSTRACT:
A state recovery and restart method that simplifies assist handling. The recovery and restart method also handles micro-branch mispredictions. An assist sequence is executed in microcode to assist an error-causing macroinstruction. If data is required from an error-causing macroinstruction, it is fetched, decoded, and macro-alias registers are restored with macro-alias data. To recover the state of the micro-alias registers, micro-alias data from a micro-operation of the flow may be loaded into the micro-alias register. Subsequently, control returns to

the Micro-operation Sequence (MS) unit to issue further error correction Control micro-operations (Cuops). In order to simplify restart, the Cuops originating from the error-causing macroinstruction supplied by the translate programmable logic arrays (XLAT PLAs) are loaded into the Cuop registers, with their valid bits unasserted. If microcode requests a restart beginning at one of the Cuops stored in the Cuop register, then the bits for that Cuop and subsequent Cuops are marked valid. Thus, the instruction can be restarted anywhere within the microcode sequence.

SUMMARY:

BSUM(7)

Computers . . . executable by the computer processor. In order to run these high level programs, the program is compiled by a compiler **program** that **translates** the higher level instructions into macroinstructions having a format that can be decoded and executed. The compiled macroinstructions are supplied. . .

DETDESC:

DETD(31)

As . . . result of extracted data, architectural machine state, or some combination of the two. Examples of alias fields include logical registers, **address size**, **data size**, stack address and stack **data size**, immediate and displacement data, branch information, and portions of various predetermined opcodes.

DETDESC:

DETD(32)

The . . . base field 272, a register index field 274, an immediate data register 276, a displacement data register 278, a stack **address size** field 280, a **data size** field 284 which indicates the size of the data as well as some floating point information, and an **address size** field 286 that indicates whether the data address is 16- or 32-bits.

7. 5,537,629, Jul. 16, 1996, Decoder for single cycle decoding of single prefixes in variable length instructions; Gary L. Brown, et al., 395/386; 364/255.3, 259.9, 262.81, DIG.1 :IMAGE AVAILABLE:

US PAT NO: 5,537,629 :IMAGE AVAILABLE: L13: 7 of 9

ABSTRACT:
A prefix decoder for decoding a plurality of prefixes of a variable length instruction code, in order to supply multiple prefix vectors to a multiple instruction decoder without incurring a one clock penalty. The parallel prefix decoder includes a plurality of prefix decoders, each coupled to receive an instruction byte from an instruction buffer, and in response thereto to supply a prefix vector that includes coded prefix information in a format that is easy to use by subsequent decoder logic. A multiplexer receives the plurality of prefix vectors, and if a steered macroinstruction has a single prefix byte, then a control circuit selects the prefix vector to supply to the macroinstruction decoder. If multiple macroinstructions are steered to multiple macroinstruction decoders, then a prefix vector can be supplied to each decoder.

SUMMARY:

BSUM(7)

Computers . . . executable by the computer processor. In order to run

these high level programs, the program is compiled by a compiler **program** that **translates** the higher level instructions into macroinstructions having a format that can be decoded and executed. The compiled macroinstructions are supplied. . .

DETDESC:

DETD(31)

As . . . result of extracted data, architectural machine state, or some combination of the two. Examples of alias fields include logical registers, **address size**, **data size**, stack address and stack **data size**, immediate and displacement data, branch information, and portions of various predetermined opcodes.

DETDESC:

DETD(32)

The . . . base field 272, a register index field 274, an immediate data register 276, a displacement data register 278, a stack **address size** field 280, a **data size** field 284 which indicates the size of the data as well as some floating point information, and an **address size** field 286 that indicates whether the data address is 16- or 32-bits.

3. 5,630,083, May 13, 1997, Decoder for decoding multiple instructions in parallel; Adrian L. Carbine, et al., 395/388; 364/259.9, 262.4, 262.7, 262.8, DIG.1; 395/387 :IMAGE AVAILABLE:

US PAT NO: 5,630,083 :IMAGE AVAILABLE: L13: 3 of 9

ABSTRACT:
A decoder for decoding multiple instructions in parallel, including a full decoder that can decode an instruction into multiple micro-operations, and a partial decoder that can decode a subset of the full instruction set.

SUMMARY:

BSUM(6)

Computers . . . executable by the computer processor. In order to run these high level programs, the program is compiled by a compiler **program** that **translates** the higher level instructions into macroinstructions having a format that can be decoded and executed. The compiled macroinstructions are supplied. . .

DETDESC:

DETD(31)

As . . . result of extracted data, architectural machine state, or some combination of the two. Examples of alias fields include logical registers, **address size**, **data size**, stack address and stack **data size**, immediate and displacement data, branch information, and portions of various predetermined opcodes.

DETDESC:

DETD(32)

The . . . base field 272, a register index field 274, an immediate data register 276, a displacement data register 278, a stack **address size** field 280, a **data size** field 284 which indicates the size of the data as well as some floating point information, and an **address size** field 286 that indicates whether the data address is 16- or 32-bits.

4. 5,566,298, Oct. 15, 1996, Method for state recovery during assist and restart in a decoder having an alias mechanism; Darrell D. Boggs, et al., 395/182.08, 182.13, 185.02 :IMAGE AVAILABLE:

US PAT NO: 5,566,298 :IMAGE AVAILABLE: L13: 4 of 9

ABSTRACT:
A state recovery and restart method that simplifies assist handling. The recovery and restart method also handles micro-branch mispredictions. An assist sequence is executed in microcode to assist an error-causing macroinstruction. If data is required from an error-causing macroinstruction, it is fetched, decoded, and macro-alias registers are restored with macro-alias data. To recover the state of the micro-alias registers, micro-alias data from a micro-operation of the flow may be loaded into the micro-alias register. Subsequently, control returns to

the Micro-operation Sequence (MS) unit to issue further error correction Control micro-operations (Cuops). In order to simplify restart, the Cuops originating from the error-causing macroinstruction supplied by the translate programmable logic arrays (XLAT PLAs) are loaded into the Cuop registers, with their valid bits unasserted. If microcode requests a restart beginning at one of the Cuops stored in the Cuop register, then the bits for that Cuop and subsequent Cuops are marked valid. Thus, the instruction can be restarted anywhere within the microcode sequence.

SUMMARY:

BSUM(7)

Computers . . . executable by the computer processor. In order to run these high level programs, the program is compiled by a compiler **program** that **translates** the higher level instructions into macroinstructions having a format that can be decoded and executed. The compiled macroinstructions are supplied. . .

DETDESC:

DETD(31)

As . . . result of extracted data, architectural machine state, or some combination of the two. Examples of alias fields include logical registers, **address size**, **data size**, stack address and stack **data size**, immediate and displacement data, branch information, and portions of various predetermined opcodes.

DETDESC:

DETD(32)

The . . . base field 272, a register index field 274, an immediate data register 276, a displacement data register 278, a stack **address size** field 280, a **data size** field 284 which indicates the size of the data as well as some floating point information, and an **address size** field 286 that indicates whether the data address is 16- or 32-bits.

7. 5,537,629, Jul. 16, 1996, Decoder for single cycle decoding of single prefixes in variable length instructions; Gary L. Brown, et al., 395/386; 364/255.3, 259.9, 262.81, DIG.1 :IMAGE AVAILABLE:

US PAT NO: 5,537,629 :IMAGE AVAILABLE: L13: 7 of 9

ABSTRACT:
A prefix decoder for decoding a plurality of prefixes of a variable length instruction code, in order to supply multiple prefix vectors to a multiple instruction decoder without incurring a one clock penalty. The parallel prefix decoder includes a plurality of prefix decoders, each coupled to receive an instruction byte from an instruction buffer, and in response thereto to supply a prefix vector that includes coded prefix information in a format that is easy to use by subsequent decoder logic. A multiplexer receives the plurality of prefix vectors, and if a steered macroinstruction has a single prefix byte, then a control circuit selects the prefix vector to supply to the macroinstruction decoder. If multiple macroinstructions are steered to multiple macroinstruction decoders, then a prefix vector can be supplied to each decoder.

SUMMARY:

BSUM(7)

Computers . . . executable by the computer processor. In order to run

these high level programs, the program is compiled by a compiler **program** that **translat** the higher level instructions to macroinstructions having a format that can be decoded and executed. The compiled macroinstructions are supplied. . .

DETDESC:

DETD(31)

As . . . result of extracted data, architectural machine state, or some combination of the two. Examples of alias fields include logical registers, **address size**, **data size**, stack address and stack **data size**, immediate and displacement data, branch information, and portions of various predetermined opcodes.

DETDESC:

DETD(32)

The . . . base field 272, a register index field 274, an immediate data register 276, a displacement data register 278, a stack **address size** field 280, a **data size** field 284 which indicates the size of the data as well as some floating point information, and an **address size** field 286 that indicates whether the data address is 16- or 32-bits.